**Leandro Henrique Lima e Silva**

**Millena Gomes da Costa**

**PARADIGMA DE DIVISÃO E CONQUISTA**

**Leandro Henrique Lima e Silva**

**Millena Gomes da Costa**

**PARADIGMA DE DIVISÃO E CONQUISTA**

Trabalho de Algoritmos e Técnicas de Programação do Curso de Ciência da Computação na Universidade Estadual Paulista – Unesp, Faculdade de Ciências e Tecnologia - FCT, orientado pelo Professor Marco Piteri, como requisito parcial para obtenção de nota e conhecimento.

**SUMARIO**

**1.PRINCIPIO DE DIVIDIR E CONQUISTAR**

**2.DETALHAMENTO DO CÓDIGO DE QUICKSORT**

**2.1 A função de troca**

**2.2 A função calcularMediana**

**2.3 A função Partition**

**2.4 A função quicksortMedianaDeTres**

**2.5 As funções em Pseudocódigo**

**3.** **ANALISE DAS DIFERENTES OPÇÕES DE ESCOLHA DO PIVÔ**

**3.1 O primeiro elemento**

**3.2 O último elemento**

**3.3 O elemento do meio**

**3.4 Mediana de três**

**3.5 Escolha aleatória**

**4.COMENTÁRIOS GERAIS**

**5.CONCLUSÕES**

**6.REFERÊNCIAS**

**1.PRINCIPIO DE DIVIDIR E CONQUISTAR**

O principio da divisão e conquista é um método, do qual sua entrada é ramificada diversas vez, a fim de resolver problemas grandes por meio de uma quebra de problemas maiores em menores (ou subproblemas) do mesmo tipo, esta iteração é realizada por meio de recorrência, na qual a estratégia se repete até que a solução seja direta ou trivial.

A técnica utilizada é separada em três etapas bem estabelecidas: dividir, conquistar e combinar.

Divisão: Na fase da divisão o algoritmo por meio de recursão, divide o problema primitivo até que, a instância fique muito menor, e fácil de ser solucionada. Considerando uma questão com tamanho n, após o uso do método, esta divisão reincide até que n seja igual a 1, 2 ou 3, ou seja, simples de se obter uma solução.

Conquista: A partir do problema base encontrado pela divisão, ou seja, quando for suficientemente pequeno e não for mais preciso dividir, então a fase de conquista resolve os subproblemas por meio de recursão, transferindo os resultados para o subproblema mais acima, até que o problema seja solucionado.

Combinar: Na fase da combinação, é utilizado um método de união dos subproblemas obtidos na fase da conquista, e então é aplicado a estratégia da conquista novamente, até que encontre a solução do problema primitivo.

A figura abstrata faz referência ao princípio dividir e conquistar. Nele é visto o problema sendo dividido em subproblemas até que se torne facilmente resolvível. Diante disso, ele retorna por meio da recursão realizando os subproblemas e combinando estes, até que se tenha uma solução definitiva.

**Vantagens**

* Indicado para aplicações que tem restrição de tempo.
* É de fácil implementação.
* Simplifica problemas complexos.
* Solução “direta” do ponto de vista da recursividade;
* Elegância;
* Usualmente produz algoritmos eficientes (complexidade logarítmica!!!)

**Desvantagens**

* Necessidade de memória auxiliar.
* Repetição de Subproblemas.
* Tamanho da pilha (número de chamadas recursivas e/ou armazenadas pode causar estouro de memória).

**2.Detalhamento do Código de QuickSort**

O código QuickSort pode ser realizado de diversas maneiras, assim como visto acima. Portanto, o método utilizado pela equipe foi o QuickSort com a estratégia de mediana de três, devido ser mais eficiente, mesmo em seus piores casos.

**2.1 A função de troca**

troca(inteiro A[], inteiro i, inteiro j)

| inteiro temp ← A[i];

| A[i] ← A[j];

| A[j] ← temp;

fim troca

Esta função, é responsável por realizar a troca entre os elementos do Array. Ela é utilizada após a comparação de um determinado componente com o Pivô que está sendo referenciado nesta interação.

**2.2 A função calcularMediana**

Esta função, a princípio, é iniciada três variáveis (a, b e c) que armazena o primeiro, ultimo e o elemento do meio do conjunto. Em seguida, com uma sequência de if’s é realizado uma operação para definir qual elemento entre os três está na posição central, ou seja, um calculo de mediana que será enviada a função Partition.

calcularMediana(inteiro A[], inteiro inicio, inteiro fim)

| inteiro meio ← (inicio + fim)/2;

| inteiro a ← A[inicio];

| inteiro b ← A[meio];

| inteiro c ← A[fim];

| inteiro medianaIndice ← 0;

| se(a < b)

| | se(b < c)

| | | medianaIndice ← meio;

| | senão

| | | se(a < c)

| | | | medianaIndice ← fim;

| | | senão

| | | | medianaIndice ← inicio;

| | | fim\_se

| | fim\_se

| senão

| | se(c < b)

| | | medianaIndice ← meio;

| | senão

| | | se(c < a)

| | | | medianaIndice ← fim;

| | | senão

| | | | medianaIndice ← inicio;

| | | fim\_se

| | fim\_se

| fim\_se

| retorne medianaIndice;

fim calcularMediana

**2.3 A função partition.**

A partir destes procedimentos é adicionado a mediana no fim do conjunto para que possa ser utilizado finalmente o método QuickSort de Cormen, no qual o pivô é o último elemento.

particao(inteiro A[], inteiro inicio, inteiro fim)

|

| inteiro medianaIndice ← calcularMediana(A,inicio,fim);

|

| trocar(A, medianaIndice, fim);

|

| inteiro pivo ← A[fim];

| inteiro i ← inicio – 1;

| inteiro j;

|

| para j ← inicio até fim - 1

| | se(A[j] <= pivo)

| | | i ← i + 1;

| | | trocar(A, i, j);

| | fim\_se

| fim\_para

| trocar(A, i + 1, fim);

| retorne i + 1;

fim\_particao

Indo adiante na função, utiliza-se métodos de comparação, no qual o pivô passa a ocupar a posição correta, isto é, a posição onde ele deve estar ao final da ordenação.

Com a utilização do método for, é iniciado a execução na qual, o pivô que ocupa agora a ultima posição passa a ser comparado, com a utilização dos índices i e j.

Assim, o índice j é associado ao primeiro elemento, e durante as interações é realizado uma comparação entre cada elemento e o pivô. Se este elemento for menor, ele realiza um troca, na qual o elemento passa a associar a posição i e então é incrementado i. Deste modo após isso, todos os elementos que forem menores que o pivô, passam a estar juntos a esquerda, e os maiores a direita. Ao final, fora do laço (*loop*), trocamos o pivô com ***i*** + 1, que passa a tomar sua posição definitiva.

**2.4 A função quicksortMedianaDeTres.**

quicksortMedianaDeTres(inteiro A[],inteiro inicio, inteiro fim)

| se(inicio < fim)

| | inteiro q = particao(A, inicio, fim) //realiza a partição

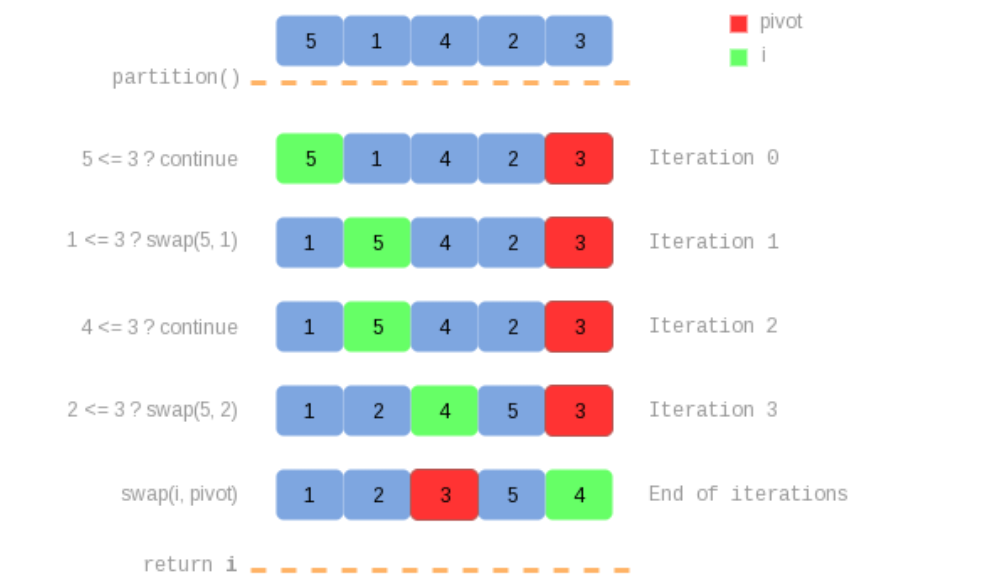
| | quicksortMedianaDeTres(A, inicio, q - 1) /ordena a partição esquerda

| | quicksortMedianaDeTres(A, q + 1, fim) //ordena a partição direita

| fim\_se

fim\_quicksort

E por último, utiliza-se a função do QuickSort com o método de divisão e conquista, no qual é definida a posição ordenada do pivô e então é realizado as mesmas operações em cada lado do conjunto, recursivamente.



A imagem demonstrada, ilustra como funciona o método do QuickSort a partir do instante em que é definido o pivô no último elemento do conjunto.

**3.** **ANALISE DAS DIFERENTES OPÇÕES DE ESCOLHA DO PIVÔ**

O pivô tema a função de ser o elemento de comparação, para que a array seja particionada de modo, que todos os elementos que estão à esquerda do pivô sejam menores ou iguais que ele e os da direita sejam maiores que ele. Alguns métodos foram estudados para que se escolha o melhor pivô, o método precisa ser simples para um bom desempenho o algoritmo e ao mesmo tempo ter uma boa chance de evitar os valores extremos (o maior ou o menor elemento da array). Entre os métodos mais utilizados estão: o primeiro elemento, o último elemento, o elemento do meio, mediana de três e a escolha aleatória.

* 1. **O primeiro elemento**

Essa técnica consiste em pegar o primeiro elemento da array e comparar esse elemento com todos os ouros elementos do vetor. O pior caso desse método é quando os métodos estão em ordem decrescente ou crescente.

* 1. **O último elemento**

Esse método consiste em pegar o último elemento do vetor e comparar ele com o restante da array. No pior caso dessa técnica é quando os métodos estão em ordem crescente ou decrescente.

* 1. **O elemento do médio**

Esse método consiste em pegar o comprimento da array e dividir por n/2 se o comprimento da array for ímpar o pivô será a parte inteira da divisão de n/2 mais um. Essa técnica intuitivamente poderia sem uma boa escolha, mas, não funciona bem em alguns casos, levando o algoritmo a complexidades O(n2).

* 1. **Mediana de três**

O método consiste em escolher três elementos da array e tomar como pivô a mediana desses três elementos. Em geral os elementos escolhidos são o primeiro, o do meio e o último elemento do vetor para fazer o cálculo da mediana. É importante lembrar que essa estratégia é recomendada apenas em vetores grandes, pois obter três elementos aleatórios e calcular a mediana deles é mais custoso para um se fazer com vetores pequenos. Com essa estratégia é esperado um tempo de execução de O(n ⋅ log ⋅ n), o que garante agilidade no processo e possui altas taxas de sucesso, outra vantagem de usar esse método é que o  particionamento será balanceado mesmo quando o vetor estiver ordenado em ordem crescente ou decrescente. mas, sempre é possível encontrar um caso no qual a complexidade é O(n2).

* 1. **Escolha aleatória**

Usar o método da escolha aleatória consiste em escolher um elemento aleatoriamente do vetor como o pivô e trocar ele com o elemento da última posição da array. O vetor será subdividido em vetores menores de em medida ¼ e ¾ de tamanho. Adotando essa técnica de pivô o tempo esperado de execução de O(n ⋅ log ⋅ n), mas, ele não recomentado pós seu comportamento é imprevisível. Assim como nos outros métodos citados o método da escolha aleatória também pode ser encontrado casos raros de complexidade de O(n2).

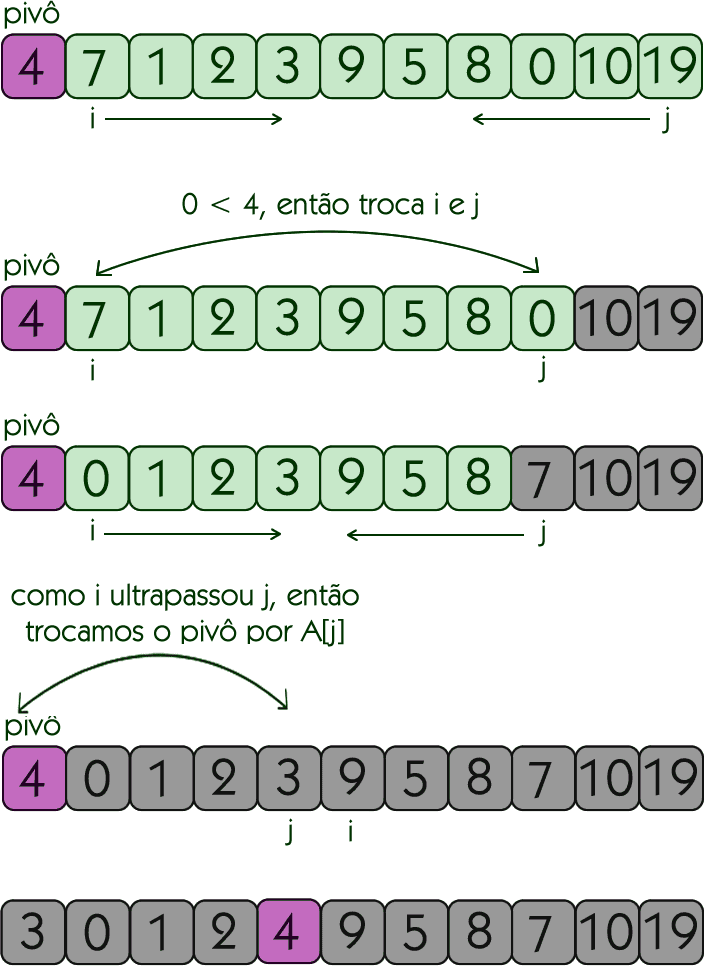
### 4.Comentários Gerais

#### **3.1 Os métodos QuickSort de Hoare e Cormen**

O que difere estes métodos, é a escolha do pivo e como este vai ser comparado com os elementos, de modo a ocupar sua posição correta quando o conjunto estiver ordenado.

O projeto a ser desenvolvido pela equipe, está sendo com base no QuickSort Cormen, no qual o pivo é referenciado como o ultimo elemento do Array, e então comparado com os índices i e j, que caminham da direita para a esquerda.

No entanto, alguns vídeos, figuras ilustrativas e animações adicionadas no trabalho, utilizam o método de Hoare, que tem como pivo o primeiro elemento do conjunto. E então os índices i e j caminham em direções opostas realizando comparações, que definiram os elementos menores do que o pivo a esquerda, e os maiores a direita.



A imagem ilustrada, demonstra claramente a estratégia de Hoare, em que a interação só termina quando o índice i ultrapassa o j, e então chama-se a função troca para tornar o pivo o índice j.

**3.2 Materiais complementares**

# [BLOG CYBERINi](https://www.youtube.com/channel/UCRD_6Vp8VEuU_IoZHBUrkiw) - Animação do QuickSort (ordenação rápida)

2018. Disponível em: <https://www.youtube.com/watch?v=h3nsF8laVi4>. Acesso em: 14 out. 2019

# [ALGORYTHMICS](https://www.youtube.com/channel/UCIqiLefbVHsOAXDAxQJH7Xw) - Quick-sort with Hungarian (Küküllőmenti legényes) folk dance 2011. Disponível em: <<https://www.youtube.com/watch?v=ywWBy6J5gz8&t=291s>>. Acesso em: 14 out. 2019

# [Algorithm guru](https://www.youtube.com/channel/UC09dmYCsQ-pHfd8i1G1rQ5Q) - QuickSort exemple 2017. Disponível em: <<https://www.youtube.com/watch?v=ywWBy6J5gz8&t=291s>>. Acesso em: 14 out. 2019

**Referências**

CORMEN, Thomas; BALKCOM, Devin. **Algoritmos de divisão e conquista.** Disponível em: <<https://pt.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms> >

CORMEN, Thomas; BALKCOM, Devin. **Visão geral do quicksort.** Disponível em: <<https://pt.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort>>

S/A. **Divisão e conquista.** Disponível em: <<https://pt.wikipedia.org/wiki/Divis%C3%A3o_e_conquista>>

FELIPE, Henrique. **Quicksort (análise e implementações).** Disponível em: <<https://www.blogcyberini.com/2018/08/quicksort-analise-e-implementacoes.html>>

FELIPE, Henrique. **Quicksort mediana de três.** Disponível em: <<https://www.blogcyberini.com/2018/08/quicksort-mediana-de-tres.html>>

FELIPE, Henrique. **Quicksort com pivô aleatório.** Disponível em: <<https://www.blogcyberini.com/2018/09/quicksort-com-pivo-aleatorio.html?utm_source=BlogCyberini&utm_medium=PostRecomendado&utm_campaign=InternalLinksRef>>

TOLEDO, Claudio F. M; ARANTES, Jesimar da S. **Quick Sort**: Considerações Sobre Algoritmos de Ordenação. 2017. 36 slides. Disponível em: <<https://edisciplinas.usp.br/pluginfile.php/4123108/mod_resource/content/1/ICC2_Aula_QuickSort.pdf>>

FABRO, João Alberto. **Técnicas de Projeto de Algoritmos Dividir e Conquistar.** 83 slides. Disponível em: <<http://www.dainf.ct.utfpr.edu.br/~fabro/IF64C/Dividir_e_conquistar+Busca.pdf>>

CARNEIRO, Francisco Vando Moreira; RODRIGUES, Gerardo Valdisio Viana. **Técnicas de Divisão e Conquista e de Programação Dinâmica para a resolução de Problemas de Otimização.** Revista Científica da Faculdade Lourenço Filho - v.8, n.1, Ceará 2011. Disponível em: <<http://www.pucrs.br/ciencias/viali/graduacao/po_2/literatura/pdinamica/artigos/Vol8_Artigo1.pdf>>

COELHO, Hebert; FÉLIX, Nádia. **Métodos de Ordenação**: Quick, Radix, Couting, Bucket (Sort). 85 slides. Disponível em: <<http://www.inf.ufg.br/~hebert/disc/aed1/AED1_05_ordenacao2.pdf>>

Francisco, Jairo de Souza. **QuickSort.** 2009. 50 slides. Disponível em: <<http://www.ufjf.br/jairo_souza/files/2009/12/2-Ordena%C3%A7%C3%A3o-QuickSort.pdf>>